Restart Schedules for Ensembles of Problem Instances

Matthew Streeter\textsuperscript{1}  Daniel Golovin\textsuperscript{1}  Stephen F. Smith\textsuperscript{2}

Carnegie Mellon University
\textsuperscript{1}Computer Science Department
\textsuperscript{2}The Robotics Institute

AAAI
July 24, 2007
Restart schedules

• A Las Vegas algorithm $A$ always returns a correct yes/no answer, but running time depends on random seed. Behavior of $A$ on instance $x$ can be represented as a run length distribution (RLD):

• A restart schedule is a sequence $\langle t_1, t_2, \ldots \rangle$ of integers, meaning “run $A$ for time $t_1$; if it doesn’t return an answer then restart and run for time $t_2$, ...”
Restarts and heavy tails

- Algorithms based on chronological backtracking often exhibit heavy-tailed RLDs (Gomes et al. 1998)
- Restart schedules can improve performance by orders of magnitude
Previous work

• Luby et al. (1993) considered solving a single instance with unknown RLD, and gave a *universal restart schedule* with optimal competitive ratio

• Gomes et al. (1998) showed that restart schedules could improve performance of a then state-of-the-art SAT solver

• Kautz et al. (2002) use features to predict RLD

• Ruan et al. (2002) show how to compute optimal schedule when there are $k$ distinct RLDs, but running time is exponential in $k$

• Gagliolo et al. (2007) used multi-armed bandit solver to select restart schedules online
RLDs vary across instances

- Here are RLDs for two SAT instances created by SatPlan when solving the logistics.d planning benchmark
- Restart helps in both cases
- Optimal schedule for average of two RLDs performs poorly
This talk

• **Goal:** efficiently construct a *single* restart schedule to use in solving a series of problem instances, each with a different RLD

• We consider three settings:
  
  • **Offline:** given a set of instances with known RLDs, compute an optimal restart schedule
  
  • **Learning-theoretic:** PAC-learn an optimal restart schedule from training instances
  
  • **Online:** you are fed an *arbitrary* sequence of instances, and must solve each instance before moving on to the next
The offline setting

• Given a set of RLDs, want to compute schedule that minimizes $E[\text{total CPU time}]$

• Assume CPU time for any instance capped at $B$

• We think this problem is NP-hard
Quasi-polynomial time approximation scheme

• Can get $\alpha^2$ approximation to best schedule in time $O(n(\log_\alpha B) B^{\log_\alpha \log_\alpha B})$

• Uses shortest path formulation (generalization of algorithm from last talk)
Greedy approximation algorithm

• Algorithm: Greedily append run of length $t$ to schedule, where $t$ is chosen to maximize $E[\#(\text{new instances solved})/t]$
Greedy approximation algorithm

- Algorithm: Greedily append run of length $t$ to schedule, where $t$ is chosen to maximize $E[\#(\text{new instances solved})/t]$

<table>
<thead>
<tr>
<th>Instance $x_1$</th>
<th>Instance $x_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>time$(A,x_1) = 10$</td>
<td>time$(A,x_2) = 1$ w/prob. $1/100$ $\infty$ w/prob. $99/100$</td>
</tr>
</tbody>
</table>

$S = \langle \rangle$
Greedy approximation algorithm

- Algorithm: Greedily append run of length $t$ to schedule, where $t$ is chosen to maximize $E[\#(\text{new instances solved})/t]$

<table>
<thead>
<tr>
<th>Instance $x_1$</th>
<th>Instance $x_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{time}(A,x_1) = 10$</td>
<td>$\text{time}(A,x_2) = 1$ w/prob. 1/100 $\infty$ w/prob. 99/100</td>
</tr>
</tbody>
</table>

$S = \langle 10, \rangle$
Greedy approximation algorithm

- Algorithm: Greedily append run of length $t$ to schedule, where $t$ is chosen to maximize $E[\#(\text{new instances solved})/t]$

<table>
<thead>
<tr>
<th>Instance $x_1$</th>
<th>Instance $x_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>time($A,x_1$) = 10</td>
<td>time($A,x_2$) = 1 w/ prob. 1/100</td>
</tr>
<tr>
<td></td>
<td>$\infty$ w/ prob. 99/100</td>
</tr>
</tbody>
</table>

$S = \langle 10, \rangle$
Greedy approximation algorithm

- Algorithm: Greedily append run of length $t$ to schedule, where $t$ is chosen to maximize $E[\#(\text{new instances solved})/t]$.

<table>
<thead>
<tr>
<th>Instance $x_1$</th>
<th>Instance $x_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>time($A,x_1$) = 10</td>
<td>time($A,x_2$) = 1 w/ prob. 1/100</td>
</tr>
<tr>
<td></td>
<td>$\infty$ w/ prob. 99/100</td>
</tr>
</tbody>
</table>

$S = \langle 10, 1, \infty \rangle$
Greedy approximation algorithm

• Algorithm: greedily append run of length $t$ to schedule, where $t$ is chosen to maximize $E[\#(\text{new instances solved})/t]$

<table>
<thead>
<tr>
<th>Instance $x_1$</th>
<th>Instance $x_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>time($A,x_1$) = 10</td>
<td>time($A,x_2$) = 1 w/prob. $\frac{1}{100}$, $\infty$ w/prob. $\frac{99}{100}$</td>
</tr>
</tbody>
</table>

$S = \langle 10, 1, 1, \rangle$
Greedy approximation algorithm

- Algorithm: Greedily append run of length $t$ to schedule, where $t$ is chosen to maximize $E[\#(\text{new instances solved})/t]$

<table>
<thead>
<tr>
<th>Instance $x_1$</th>
<th>Instance $x_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{time}(A,x_1) = 10$</td>
<td>$\text{time}(A,x_2) = 1$ w/prob. $1/100$</td>
</tr>
<tr>
<td></td>
<td>$\infty$ w/prob. $99/100$</td>
</tr>
</tbody>
</table>

$S = \langle 10, 1, 1, 1, \ldots \rangle$
Greedy approximation algorithm

- Algorithm: Greedily append run of length $t$ to schedule, where $t$ is chosen to maximize $E[\#(\text{new instances solved})/t]$

- Performance
  - Gives 4-approximation to optimal schedule (may do better)
  - **New variant** also returns optimal schedule if all instances have same RLD
The learning-theoretic setting

- Draw instances from some distribution. Each instance has its own RLD. Want to PAC-learn optimal restart schedule (with prob. $\geq 1-\delta$, schedule’s expected cost is $\leq \epsilon$ worse than optimal)

- Two questions:
  - how many training instances?
  - how may runs per instance?
How many training instances?

- Need $\mathcal{O}\left(\frac{B}{\epsilon^2} \sqrt{B} \log \delta^{-1}\right)$ instances, assuming RLD of each instance is known exactly.

- Proof uses shortest path formulation + Hoeffding bounds.
How many runs per instance?

- A profile \( \langle T_1, T_2, ..., T_k \rangle \) is a non-increasing list of integers.
- State of schedule \( S \) at time \( t \) can be represented as a profile \( P(S,t) \).

\[
S = \langle 1, 2, 4, ... \rangle
\]
How many runs per instance?
How many runs per instance?

- Answer: after $B$ runs, can get unbiased estimate of CPU time required by any schedule $S$
How many runs per instance?

- Answer: after $B$ runs, can get unbiased estimate of CPU time required by any schedule $S$
How many runs per instance?

- Answer: after $B$ runs, can get unbiased estimate of CPU time required by any schedule $S$
How many runs per instance?

• Answer: after $B$ runs, can get unbiased estimate of CPU time required by any schedule $S$.
How many runs per instance?

• Answer: after $B$ runs, can get unbiased estimate of CPU time required by any schedule $S$
How many runs per instance?

• Answer: after $B$ runs, can get unbiased estimate of CPU time required by any schedule $S$
How many runs per instance?

- Answer: after \( B \) runs, can get unbiased estimate of CPU time required by any schedule \( S \)
How many runs per instance?

- Answer: after $B$ runs, can get unbiased estimate of CPU time required by any schedule $S$
How many runs per instance?

- Answer: after $B$ runs, can get unbiased estimate of CPU time required by any schedule $S$
How many runs per instance?

- Answer: after $B$ runs, can get unbiased estimate of CPU time required by any schedule $S$
How many runs per instance?

- Answer: after $B$ runs, can get unbiased estimate of CPU time required by any schedule $S$
How many runs per instance?

- Answer: after $B$ runs, can get unbiased estimate of CPU time required by any schedule $S$
- If each run has time limit $B$, total CPU time $\leq B^2$
How many runs per instance?

- **Answer:** after \( B \) runs, can get unbiased estimate of CPU time required by any schedule \( S \).

- If each run has time limit \( B \), total CPU time \( \leq B^2 \).

- Can actually perform \( i \)th run with time limit \( B/i \). Total time = \( O(B \log B) \).

\[
\begin{align*}
\langle \rangle & \quad \langle 1 \rangle & \quad \langle 1,1 \rangle & \quad \langle 2,1 \rangle & \quad \langle 2,1,1 \rangle & \quad \langle 2,2,1 \rangle & \quad \langle 3,2,1 \rangle & \quad \langle 4,2,1 \rangle \\
t = 0 & \quad 1 & \quad 2 & \quad 3 & \quad 4 & \quad 5 & \quad 6 & \quad 7
\end{align*}
\]
The online setting

• World secretly selects sequence of $n$ instances/RLDs

• For $i$ from 1 to $n$
  • You select schedule $S_i$ to use to solve $i^{th}$ instance
  • As feedback you observe how much time $S_i$ takes

• $\text{regret} = \mathbb{E}[\text{your total time}] - \min_{\text{schedules } S} (\mathbb{E}[S's \text{ total time}])$
The online setting

• World secretly selects sequence of \( n \) instances/RLDs

• For \( i \) from 1 to \( n \)
  • You select schedule \( S_i \) to use to solve \( i^{th} \) instance
  • As feedback you observe how much time \( S_i \) takes

\[ \text{regret} = \mathbb{E}[\text{your total time}] - \min_{\text{schedules } S} (\mathbb{E}[S's \text{ total time}]) \]

• We give a schedule selection strategy whose worst-case regret is \( o(n) \), assuming schedules come from a small pool.

• Uses unbiased estimation procedure + technique from Cesa-Bianchi et al. (2005)

• **Ongoing work:** online version of greedy approx. algorithm
Experimental evaluation

- Ran satz-rand on formulae generated by SatPlan when solving randomly-generated logistics planning benchmarks

<table>
<thead>
<tr>
<th>Restart schedule</th>
<th>Avg. CPU (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy schedule</td>
<td>21.9</td>
</tr>
<tr>
<td>Best geometric $\langle \alpha^0, \alpha^1, \alpha^2, \ldots \rangle$</td>
<td>23.9</td>
</tr>
<tr>
<td>Best uniform $\langle t, t, t, \ldots \rangle$</td>
<td>33.9</td>
</tr>
<tr>
<td>Luby’s universal schedule</td>
<td>37.2</td>
</tr>
<tr>
<td>No restarts</td>
<td>74.1</td>
</tr>
</tbody>
</table>
Experimental evaluation

- Ran satz-rand on formulae generated by SatPlan when solving randomly-generated logistics planning benchmarks

<table>
<thead>
<tr>
<th>Restart schedule</th>
<th>Avg. CPU (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy schedule (cross-val)</td>
<td>21.9 (22.8)</td>
</tr>
<tr>
<td>Best geometric ( \langle \alpha^0, \alpha^1, \alpha^2, ... \rangle )</td>
<td>23.9</td>
</tr>
<tr>
<td>Best uniform ( \langle t, t, t, ... \rangle )</td>
<td>33.9</td>
</tr>
<tr>
<td>Luby's universal schedule</td>
<td>37.2</td>
</tr>
<tr>
<td>No restarts</td>
<td>74.1</td>
</tr>
</tbody>
</table>
Generalization: multiple Las Vegas algorithms

- If we have multiple Las Vegas algorithms, can consider restart schedules of the form $\langle (a_1, t_1), (a_2, t_2), \ldots \rangle$

- Results for all three settings generalize

- With multiple algorithms, it is NP-hard to get a $4-\epsilon$ approximation for any $\epsilon > 0$ (so greedy $4$-approx is optimal)